J Intell Robot Syst (2012) 68:165–184
DOI 10.1007/s10846-012-9683-8

A Novel Trajectory Generation Method for Robot Control

KeJun Ning · Tomas Kulvicius · Minija Tamosiunaite · Florentin Wörgötter

Received: 10 September 2011 / Accepted: 30 May 2012 / Published online: 16 June 2012

Abstract This paper presents a novel trajectory generator based on Dynamic Movement Primitives (DMP). The key ideas from the original DMP formalism are extracted, reformulated and extended from a control theoretical viewpoint. This method can generate smooth trajectories, satisfy position- and velocity boundary conditions at start- and endpoint with high precision, and follow accurately geometrical paths as desired. Paths can be complex and processed as a whole, and smooth transitions can be generated automatically. Performance is analyzed for several cases and a comparison with a spline-based trajectory generation method is provided. Results are comparable and, thus, this novel trajectory generating technology appears to be a viable alternative to the existing solutions not only for service robotics but possibly also in industry.

Keywords Trajectory generation · Dynamic trajectory joining · Control theory · Machine learning

1 Introduction

Trajectory planning is one of the most fundamental and important research fields in robotics. For a robotic manipulator, the trajectory generator is used to generate a time history of relevant variables: the desired positions, velocities, and accelerations in state space [1], i.e., joint space or task space (Cartesian space). The resulting trajectory signals are then sent to the motion control system (also known as trajectory tracker) at specific time intervals. The motion control system guarantees that the robot manipulator executes the planned trajectory [1].

Generally, the trajectory generation problem can be divided into two separate sub-problems [2]. The first one is called "path planning" and has to design path geometry by providing the sequence of points in space needed to execute a task. The second one is called "trajectory planning and optimization" and includes designing of the temporal (dynamic) structure of the trajectory based on optimization criteria. The second aspect covers

Parts of this work have been published in preliminary form at ICRA 2011 [38].

K. Ning (✉) · T. Kulvicius · M. Tamosiunaite · F. Wörgötter
Bernstein Center for Computational Neuroscience, Inst. of Physics III, University of Göttingen, 37077 Göttingen, Germany
e-mail: nkj@sjtu.org

K. Ning
Research & Technology, Lenovo, 100085 Beijing, China

many different issues and is heavily dependent on the specific application requirements.

When planning takes place in joint space, we can specify trajectories for each independent joint. A common method for joint space trajectory control uses lower order polynomials to provide interpolating points at servo rates between user-specified via-positions [1, 4]. Using one higher order polynomial to pass through all the via-points (also known as way-points) is not practical, because it will result in large oscillations of the trajectory. Since polynomials of two neighboring regions can be joined in a continuous way, we can impose specific boundary conditions (i.e., position, and higher derivatives) onto several lower order polynomials to obtain continuous trajectories. There are some other solutions for providing smooth transitions between the specified points in joint space, e.g., [4]. The problem is that in task space, one only knows the initial and goal positions of the end-effector, and can not constrain the intermediate path. Planning in joint space is popular in point-to-point applications as it is easy to implement without having to deal with the kinematics singularity problem [1].

On the other hand, when planning a trajectory in task space, we can completely specify the desired trajectory of the end-effector. As planning in task space is easier to comprehend for practical applications and can be used for completing versatile operations, it is more popular and widely studied. However, for task space planning, we need to take care of the kinematic singularity problem [1]. For most applications, we can describe the path information by a sequence of via-points in the task space. A popular approach is to utilize some simple curves (e.g., lines, arcs or parabolas) to setup a whole path in task space, depending on the assigned via-points. The problem is how to connect separate segments, in order to create a smooth path. Common solutions are based on the introduction of zones in which an interpolation between the two adjacent segments can be performed [2]. Once we get the planned path, trajectory planning and optimization can be executed, depending on the task-specific requirements and criterions.

The literature about traditional trajectory planning and optimization is substantial. Optimization topics cover energy [29, 30], jerk (the third derivative of position) [29, 31–33], time [30, 33–36], etc.; and different optimization techniques are employed (e.g., [30, 36]) as well. Generally, these approaches are complex and require physical models (dynamic parameters, specifications of the actuators, etc) of the studied robotic manipulators. Some of them also need to provide predefined path information. Mostly, these research and development activities are motivated by the increasing need for improved motion performance for achieving higher productivity in industry.

In the current paper we will address the problem of trajectory planning and generation providing a novel and versatile alternative approach that combines aspects from control engineering with a dynamic systems perspective.

In general there are two, often conflicting, demands on trajectory generation: On the one hand, trajectories should arrive at specified points in task space (start- and endpoints as well as via points) with small error - a demand usually existing for industrial robots. On the other hand, trajectories should be smooth, human-like, and robust against disturbances - a demand found in humanoid and/or service robotics, e.g., [3]. Two fundamental methods (and many variations thereof) exist, which address these demands to different degrees. In a somewhat simplified view one can broadly state that Splines (and other lower order polynomials) are very well suited for via-point control [1, 4–9], whereas trajectories generated by a dynamical systems approach (dynamic motion primitives, DMPs [10–13]) are smooth and robust against perturbations.

The goal of the current study is to augment the DMP framework such that it will obtain some of the advantages of via-point control. This paper will, thus, in the next Section 1.1 introduce DMPs so that we can discuss their pros and cons and begin developing our own framework. In Section 1.2 we will state the purpose of our investigation. In Sections 2 and 3, we present our new solution for trajectory generation. Some case studies are shown, disclosing more details and implementation issues are provided in Section 4. Section 5 will then compare our approach to others showing that the new trajectory generation algorithm presented here is a useful alternative to

the existing methods. In the Appendix we give a summary of the required parameters.

1.1 Introduction - DMPs

The DMP algorithm proposed by Schaal and Ijspeert et al. is "a formulation of movement primitives with autonomous non-linear differential equations whose time evolution creates smooth kinematic control policies" [10].

The original DMP algorithm consists of two sets of differential equations namely a canonical system $\tau\dot{x} = h(x)$ and a transformation system $\tau\dot{y} = g(y, f(x))$ [10–13]. The canonical system is used to represent the phase of the movement process. The transformation system is a basic point attractive system utilized to generate the desired movement.

For the point attractive behavior (reaching movement), Ijspeert et al. instantiated the transformation system by a second-order dynamics, as follows [13],

$$\tau\dot{z} = \alpha_z(\beta_z(g - y) - z) + f, \tag{1}$$

$$\tau\dot{y} = z, \tag{2}$$

$$f(x, v, g) = \sum_{i=1}^{N} \Psi_i w_i x / \sum_{i=1}^{N} \Psi_i \tag{3}$$

where

$$\Psi_i = \exp\left(-h_i(x - c_i)^2\right).$$

And the exponential system described by

$$\tau\dot{x} = -\alpha_x x. \tag{4}$$

In Eqs. 1–4, α_z, β_z, α_x are time constants, τ is a temporal scaling factor. y, z and $\dot{z}$ can be interpreted as desired position, velocity and acceleration, g is a given goal state. Centers c_i and widths h_i characterize the Gaussian functions ψ_i, w_i denotes weights, and the other variables are used as state variables. The phase variable x is a substitute for time, and v is a phase velocity [13].

The function f, shown in Eq. 3, is a nonlinear function approximator [14], which can be trained (by adjusting the weights w_i) to approximate discrete movements of various shapes. By appropriate parameter settings when the above-mentioned second-order system is critically damped, x acts as a "gating term" to ensure that $f = 0$ at the end of the movement. If the weights w_i in Eq. 3 are bounded, the combined system asymptotically converges to the unique point attractor g. A more detailed explanation can be found in the literature [10–13].

In order to avoid discontinuities in acceleration Eq. 1 can be replaced by

$$\tau\dot{z} = \alpha_z(\beta_z(r - y) - z) + f, \tag{5}$$

where

$$\tau\dot{r} = a_g(g - r). \tag{6}$$

The original DMP framework can be used to implement discrete as well as rhythmic control policies [15], and has exhibited great potential for learning from demonstration due to its flexibility and robustness against perturbations. Several applications using DMPs have been shown in the field of humanoid robot research. For example, tennis swings [11], reaching with obstacle avoidance ability [16, 17], constrained reaching [18], drumming [19], "Ball-in-a-Cup" game [20], hitting and batting [21], handwriting generation [22], etc.

1.2 Introduction - Goals of This Study

The basic DMP formulation has some limitations which restrict its usefulness for trajectory generation applications. For example, limited by its formulation and structure, the original DMP can not be used to directly incorporate a target velocity or a via-point (but see [21]). It always asymptotically converges to the final point and the speed of approaching the target is zero. These aspects make it not practical to chain several trajectory segments together, because a robot will have to "intermittently stop" at all the via-points. By starting the next DMP while the previous one is not totally finished one can create smooth connections of the DMPs, but this will reduce control over the accuracy of the path, and lead to a limited capability to steer the velocity at the connection points.

In the current study we will therefore modify the DMPs framework to incorporate via-point

control and to allow for the chaining of DMPs. This, way we will gain a higher level of accuracy, but we will also partially loose some of the DMPs' original properties. Specifically:

(1) The algorithm will be based on an analysis of the original DMPs from a control theoretical viewpoint considering several aspects. This is important because it extends the current literature on DMPs and allows employing conventional and well-established methods for control via second-order systems.
(2) From this, we will design a trajectory generator by modifying DMPs. The resulting algorithm is, thus, removed from the original framework. This is done to target a different application field, more related to low-level control under given, rigorous error bounds.
(3) Thus, the here presented method has its own application field (chained trajectories with precise boundary conditions and disturbance robustness). In this field it can compete (or even excel) in applicability with the original DMPs as well as with Splines and is therefore a viable alternative.
(4) The new algorithm does reintroduce strict time dependency. This is required for the above mentioned target application fields where high-accuracy along the trajectory is needed and obtained by introducing explicit timing-laws [1].

2 The Trajectory Generator Based on DMP

In this section, we will start with the task description and then present detailed information on our method.

2.1 Specification Goals for the Trajectory Generator

Figure 1a shows a schematic plan of an example task. The robot has to move from point A to B along the straight line AB, but the initial and final velocity vectors are not zero. Obviously the actuators of a robot would have to provide extremely high and unacceptable torques at the two ends. Figure 1b shows the solution of a popular polynomial-based method, which can provide a smooth and safe trajectory but remains ill-defined at intermediate positions. Figure 1c shows a natural and smooth solution: at the two connection points, transitions zones are introduced to satisfy the specified boundary conditions and to smoothen the response of the physical system. Generally, traditional solutions need to insert intermediate points and fit the zones by splines/polynomials [1], or simple curves (e.g., for industrial applications, circular arcs are employed). In this paper, as the case shown in Fig. 1c, we target to attain smooth transitions using our newly developed method. In the middle of the path, this trajectory should match the desired pathway and velocity profile. In the two end zones the trajectory has transition segments. The segments should arrive at the target point with the assigned velocity vector (orientation and magnitude) as accurately as possible. Also, the trajectory has to be smooth (i.e., the velocity change of the whole process must be continuous). A suitable compensation introduced during the transition segments will reduce jerk, leading to smooth movement of a real robotic manipulator.

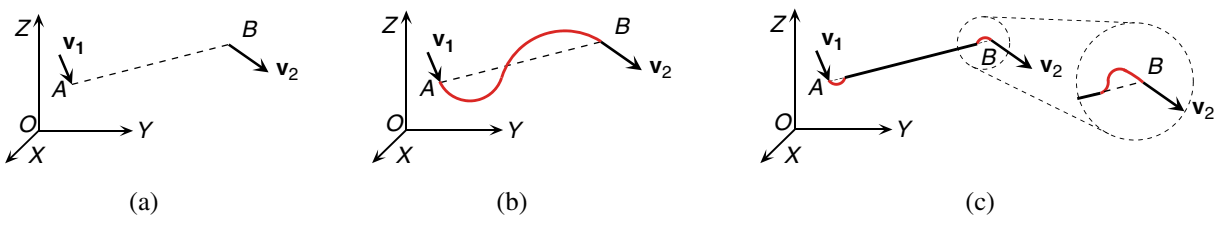

Fig. 1 A schematic plan of a task. **a** A robot needs to move from point A to point B, with the assigned velocities $\mathbf{v}_1$ and $\mathbf{v}_2$. Obviously, a straight line (*dashed*) is not practical for a robot platform. **b** A smooth trajectory can be implemented like the solid one, but intermediate positions remain undefined. **c** This track shows a smooth and acceptable solution, and is everywhere defined

In summary, the following requirements are fundamental for a complete and useful trajectory generating technology.

(1) Accurate adherence to boundary conditions in position and velocity of the start and endpoints.
(2) Global smoothness (e.g., C2: continuity of the second derivative over the closed interval of the data set) and accuracy of the trajectory.
(3) Allowing for desired speed profiles and path constraints.
(4) Definite end-time; an aspect which is important for most applications.

From such trajectories, we can build a complicated path by a serial connection of several sub-trajectories with boundary conditions for the joining between them. Furthermore, for possible industrial applications, we also need to incorporate the required velocity profile into the trajectory.

2.2 Architecture of Our Trajectory Generator

Figure 2a shows the architecture of the DMP based trajectory generator, which consists of four key modules. The Second-Order System module provides the trajectory $y(t)$. The outputs from the Boundary Function Generator $r(t)$, and Modulation Function $f(t)$, are fed into the Second-Order System. The Boundary Function Generator imposes the boundary conditions onto the system. The Modulation Function consisting of a Gaussian Kernel Based Approximating Function and a Suppressing Window Function (see Fig. 2b) will affect the Second-Order System's response as a forcing input. The Gaussian Kernel Based Approximating Function contains a weight vector, and these weights need to be trained in order to encode the information brought from the sample trajectory $y^*(t)$, provided by the Sample Trajectory Generator. After training, the Modulation Function module will contain the main information obtained from $y^*(t)$, and the sample trajectory will not be used anymore.

The architecture shown in Fig. 2 is for one Degree-of-Freedom (DOF). For an N-DOF application, we can employ N copies of this architecture in parallel.

2.3 Second-Order System Module

In the original DMP and the following studies [10–13, 15–22], parameters α_z and β_z shown in Eqs. 1 and 5 have not been directly related to the physical meaning of a second-order system. In the current paper, we adopt a standard formulation to make the system easier to understand from a control theory perspective.

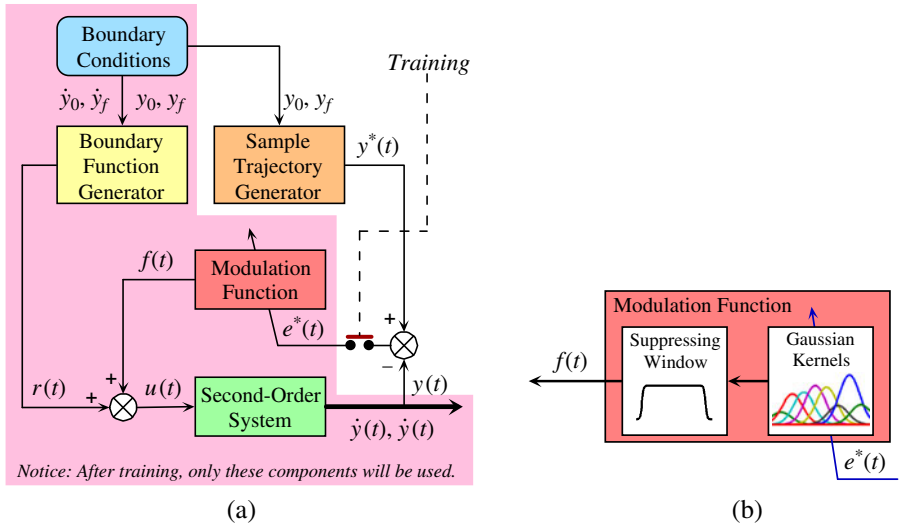

Fig. 2 The architecture of our DMP based trajectory generator. **a** The whole structure. After training, only the pink-shaded components will be used. **b** The inner components showing the Modulation Function

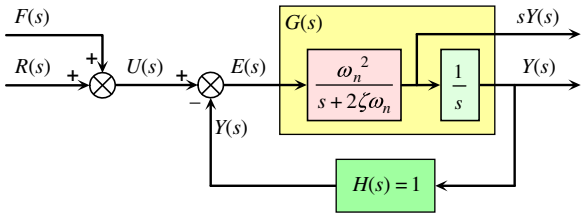

Fig. 3 Block diagrams of the employed Second-Order System in Laplace space. We can get position and velocity outputs directly

Figure 3 shows the block diagram of the Second-Order System module employed in this paper, expressed in Laplace space. Its forward channel is $G(s) = \frac{\omega_n^2}{s(s+2\zeta\omega_n)}$, and its feedback channel is $H(s) = 1$. It is a type-1 system. Let $U(s) = R(s) + F(s)$ and $Y(s)$ denote the input and output respectively, then the transfer function is as follows,

$$\frac{Y(s)}{U(s)} = \frac{G(s)}{1 + G(s)\,H(s)} = \frac{\omega_n^2}{s^2 + 2\zeta\omega_n s + \omega_n^2}. \quad (7)$$

This is a standard second-order system, where the constants ζ and ω_n are the damping coefficient and the natural undamped frequency of the system, respectively. As we know, when $\zeta = 1$, the system has a double pole at $-\omega_n$, and this results in a critically damped response [23].

Let us define $y_2(t) = \dot{y}_1(t) = \dot{y}(t)$, then in state space the Second-Order System is described as,

$$\dot{\mathbf{Y}} = \mathbf{A}\mathbf{Y} + \mathbf{B}\mathbf{u}, \quad (8)$$

where, $\mathbf{Y} = \begin{bmatrix} y_1(t) \\ y_2(t) \end{bmatrix}$, $\mathbf{A} = \begin{bmatrix} 0 & 1 \\ -\omega_n^2 & -2\zeta\omega_n \end{bmatrix}$, $\mathbf{B} = \begin{bmatrix} 0 \\ \omega_n^2 \end{bmatrix}$, $\mathbf{u} = u(t) = r(t) + f(t)$.

The model shown in Eq. 7 is well-studied in control theory. It is well known that its steady-state error can be calculated by the final-value theorem [23], as follows,

$$e_{ss} = \lim_{t\to\infty} e(t) = \lim_{s\to 0} \frac{sU(s)}{1 + G(s)} \quad (9)$$

Furthermore, the error's change rate can be expressed as

$$\dot{e}_{ss} = \lim_{t\to\infty} \dot{e}(t) = \lim_{s\to 0} \frac{s^2 U(s)}{1 + G(s)} \quad (10)$$

Spreading Eq. 8 we can get the relation equations relating ζ and ω_n to the parameters α_z and β_z, shown in Eqs. 1, 5 as follows,

$$\omega_n = \frac{\sqrt{\alpha_z \beta_z}}{\tau}, \quad (11)$$

and,

$$\zeta = \frac{1}{2}\sqrt{\frac{\alpha_z}{\beta z}}. \quad (12)$$

Schaal et al. [10] only provided $\beta_z = \alpha_z/4$ to get critical damping and had not been concerned with other possible frequency characteristics and responses of their DMPs. Entering this equation into Eq. 12 we can get $\zeta = 1$ (criterion for critical damping). Please note, linear and nonlinear subsystems are interconnected in the DMP method proposed. The Second-Order System shown as Eq. 7 is positive real and passive. An interconnection containing a passive subsystem (linear or not) with a strictly proper, strictly positive real one, is always closed-loop stable [39–42]. Furthermore, based on Eqs. 9 and 10 and the time-domain response of such a system [23], we can explain why the original DMPs always asymptotically converge to the final point and the speed of approaching the target is zero. Equations 9 and 10 also provide the foundation for including the boundary conditions in the architecture shown in Fig. 2. We will discuss this issue in the following sections.

2.4 Boundary Function Generator

For a trajectory generator, we have to take into account the boundary conditions (i.e., position, velocity). As mentioned above, polynomials are normally employed to remove discontinuities in velocity and acceleration between adjacent path segments. In order to allow imposing continuity of velocities at the junction points, the lowest order for an interpolating polynomial is three (also known as cubic polynomial, in Eq. 13, $N_P = 3$) [1]. For an application with acceleration constraints, fifth-order polynomials can be employed.

$$x(t) = \sum_{j=0}^{N_p} a_j t^j \quad (13)$$

We use a third-order polynomial and determine the coefficients given by y_0, y_f and $\dot{y}_0$, $\dot{y}_f$ as the positions and velocities at the start- and endpoints respectively.

However, we cannot employ the third-order polynomial alone to construct the Boundary Function Generator shown in Fig. 2. As we know, a type-1 system cannot follow a parabolic or a higher order function, because the steady-state error is infinite [23]. This conclusion can be derived from Eq. 9. Next we will show how we can fix this problem. If $u(t)$ is a ramp function, from Eqs. 9 and 10 we obtain

$$e_{ss} = V/K_v, \tag{14}$$

and

$$\dot{e}_{ss} = 0, \tag{15}$$

where V is the slope coefficient of $u(t)$, and $K_v = \omega_n/2\zeta$. Equation 15 shows that the slope coefficient of the Second-Order System's response will approach that of the input ramp function in the end. Even though Eqs. 14 and 15 hold only for the steady-state, we can design our system based on this principle with controllable precision. In fact, Eq. 15 is the proof for approaching the assigned velocity and Eq. 14 presents the offset we can use to reduce the position error. Thus, we can utilize a third-order polynomial extended by a line segment to construct the reference signal $r(t)$. Figure 4 shows such a case.

Let us denote t_m as the junction moment (the time point where the polynomial segment and the line segment are joined), then the reference signal is defined as follows:

$$r(t) = \begin{cases} a_0 + a_1 t + a_2 t^2 + a_3 t^3 & 0 \le t \le t_m \\ y_e - (t_f - t)\,\dot{y}_f & t_m < t \le t_f \end{cases} \tag{16}$$

where,

$$y_e = y_f + \delta_e \tag{17}$$

and $\delta_e = (2\zeta/\omega_n)\,\dot{y}_f$.

In Eq. 16, the polynomial segment $[0,\ t_m]$ is employed to connect the boundary conditions imposed on the start point, and the linear segment $(t_m,\ t_f]$ works as a ramp input which provides a convergence reference for the Second-Order System, with both, position and velocity, constraints. The coefficients of the polynomial segment are determined as follows,

$$\begin{cases} a_0 = y_0 \\ a_1 = \dot{y}_0 \\ a_2 = \dfrac{3\,(y_m - y_0) - t_m\,(2\dot{y}_f - \dot{y}_0)}{t_m^2} \\ a_3 = \dfrac{2\,(y_0 - y_m) + t_m\,(\dot{y}_0 + \dot{y}_f)}{t_m^3} \end{cases} \tag{18}$$

where, $y_m = y_e - (t_f - t_m)\dot{y}_f = y_f + [(2\zeta/\omega_n) - (t_f - t_m)]\dot{y}_f$. The calculation method for the coefficients of the third-order polynomial segment can be found in the literature [1]. The value t_m depends on the temporal properties of the Second-Order System module (i.e., ω_n and ζ), see later.

The Boundary Function Generator will guarantee the required system's performance at start and end regions.

2.5 Modulation Function

For most application tasks, we need to control a robot to move along an assigned path, e.g., the case shown in Fig. 1. The Modulation Function $f(t)$ shown in Fig. 2 is used to "force" the generated response to follow the assigned trajectory (i.e., the sample trajectory, see later) during the middle of the processing.

As shown in Fig. 2, the input to the Second-Order System is

$$u(t) = r(t) + f(t). \tag{19}$$

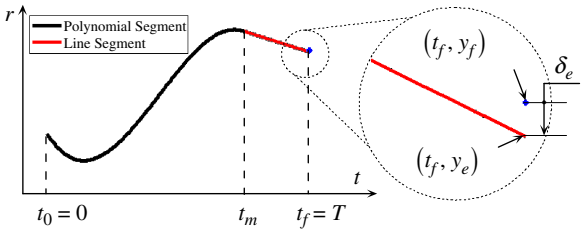

Fig. 4 A solution for building the Boundary Function Generator. A third-order polynomial extended by a line segment can be employed here to construct the reference signal $r(t)$ for the DMP based trajectory generator

The variable $f(t)$ works as the forcing input to this system and is used to steer the final result. Please note, reference signal $r(t)$ defined as Eq. 16 provides mathematical derivability (at the junction points), in the middle section, however, it is unconstrained. By introducing a suitable $f(t)$, we can totally counteract unwanted effects brought by $r(t)$ and to reshape the system's output in order to force it to follow the desired path. The Gaussian Kernels Based Approximating Function, as described below, is a solution for getting such a suitable $f(t)$.

2.5.1 Gaussian Kernels Based Approximating Function

The original DMP introduces a nonlinear control based on learned feed-forward controllers [10–13]. A nonlinear function based on a group of Gaussian kernels can be trained to smoothly approximate a given sample curve. We utilize this idea to build the Modulation Function for our system. Let us define the Modulation Function with M kernels,

$$f(t) = \frac{\mathbf{\Psi^T W}}{\sum_{i=1}^{M} \psi_i} v(t), \tag{20}$$

where $\mathbf{W} = [w_1, w_2, \cdots, w_i, \cdots, w_M]^T$, w_i is the weight of the Gaussian kernel i and $\mathbf{\Psi} = [\psi_1, \psi_2, \cdots, \psi_i, \cdots, \psi_M]^T$, with

$$\psi_i = \exp\left(-h_i\left(\frac{t}{T} - c_i\right)^2\right), \tag{21}$$

where $T = t_f - t_0$ is the time length of the whole process, t is the time, and $v(t)$ is a suppressing window function (see later). Constants c_i and h_i are centers and widths of the Gaussian kernels i, evenly distributed within the interval [0,1]. To simplify the treatment, we let $h_i = h$, $i = 1, 2, \ldots, M$.

2.5.2 Weight Learning

In this paper, the weight vector $\mathbf{W}$ will be trained to match the sample trajectory $y^*(t)$ generated by the Sample Trajectory Generator shown in Fig. 2.

Several different methods exist for weight adaptation like locally weighted regression methods [11, 24] or global regression methods [25]. Here we propose a simple and practical learning rule to train the weight vector $\mathbf{W}$. This procedure is similar to back-propagation learning used in artificial neural networks [26], where the error between target signal and system's output is used to change weights. In our case this can be formalized by the following equation:

$$\Delta w_i = \gamma \left(y^*(k) - y(k)\right) v(k), \tag{22}$$

where, γ is the learning rate, $k = c_i T$ defines the center of the *i-th* Gaussian kernel within the time period [0, T]. Here, k serves to anchor the Gaussian kernels to the time period T. By $k = c_i T$, we project M Gaussian kernels from the range [0,1] to [0, T], and therefore only need to iterate Eq. 22 at time moments $c_i T$, $i = 1, 2, \ldots, M$.

The present "simple" solution will be robust to approximate the sample trajectory and the final accuracy (in the middle section of the path) is depending on the learning rate (see Eq. 22).

Note, this simple approach might not be optimal for a single training trajectory. It is more suitable for cases where many different (but similar) training trajectories are used and this method should lead to an average trajectory. In general any other more sophisticated method (e.g., LWPR [10, 11, 24]) can be used to learn weights of the kernels.

2.5.3 Suppressing Window Function

The Suppressing Window Function $v(t)$ in Eq. 20 serves as an "Enable" term inside the Modulation Function shown in Fig. 2b. It controls kernel influence by suppressing their action near start- and endpoints thereby assuring accuracy of the process. When the system is in those regions, we should let the Boundary Function Generator's output $r(t)$ drive the Second-Order System and Eqs. 14–16 will guarantee that the assigned boundary conditions will be successfully reached. Figure 5 shows a solution for a suitable Suppressing Window Function. The double-sigmoid function is continuous and differentiable and it has

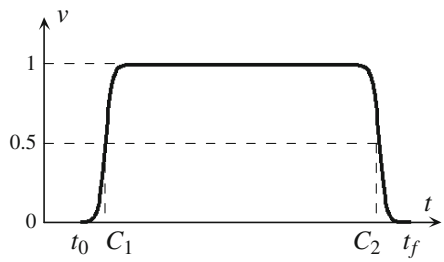

Fig. 5 Smooth suppressing window function given by the product of two sigmoids

a pair of horizontal asymptotes as $t \to \pm\infty$. We define it by,

$$v(t) = s_1(t) \times s_2(t), \tag{23}$$

with,

$$s_1(t) = \frac{1}{1 + e^{-l_1(t-C_l)}}, \text{ and } s_2(t) = \frac{1}{1 + e^{-l_2(t-C_2)}}, \tag{24}$$

where, l_1, l_2 are used to set the slopes, and C_1, C_2 to set the inflection points (see Fig. 5). With these parameters, we need to ensure that $v(t)$ approaches zero at the start- and end-time as close as possible in order to obtain $f(t) = 0$. This way, $r(t)$ totally governs the Second-Order System at the start- and endpoints and a more precise result is achieved. This kind of transition region is similar to the popular "zone solution" [1, 2], but implicitly described by the parameters of the sigmoids. Figure 1c shows the two regions at the ends of the trajectory. In the following applications cases, this aspect is analyzed in more detail. We define $l_1 = l_2 = 20$, $C_1 = 0.3$, and $C_2 = t_f - 0.38$ for all following examples. For the policy on how to choose these parameters see below.

2.6 Sample Trajectory Generator

Planning a trajectory by this method is done in two steps. In the first step the sample trajectory provides geometric path constraints and the speed profile information along the trajectory. In the second step the boundary conditions are imposed and the final smooth result is obtained through the interaction of $f(t)$ and $r(t)$. Note, when designing the sample trajectory sequence, we do not need to care about the possible initial and final velocities imposed by the boundary conditions (see the signal flow shown in Fig. 2), which simplifies this step substantially. Also, the sample $\mathbf{y}^*$ is only used during the training process (training the weight vector $\mathbf{W}$of the Gaussian Kernel Based Approximating Function, see Eqs. 20 and 22). After training, $f(t)$ will contain all information and $\mathbf{y}^*$ will not be used anymore.

Now we describe how to define the Sample Trajectory Generator required for the step 1. It provides a sample trajectory sequence $\mathbf{y}^*(t)$:

$$\mathbf{y}^*(t) = \left[y^*(0), y^*(\tau), \cdots, y^*(k\tau), \cdots, y^*(L\tau)\right]^T, \tag{25}$$

where, τ is the sampling interval, L is the whole step number, and then $L\tau = T$ is the total desired time to complete the path. We can acquire it by imitation (teaching and recording), or generate it by a simple path planner routine. Without loss of generality, we focus on the latter in this paper.

To generate a velocity profile for a sample trajectory, a simple trapezoidal timing law [1] can be employed, which is also known as LSPB (Linear Segments with Parabolic Blends) [27]. In fact, this method is more popular for planning and control in joint space. It has a "speedup - uniform motion - slowdown" speed profile. It represents a time optimal solution for actuators and is easy to implement. Here we need to take the maximal acceleration that the actuator can provide into account.

Obviously, LSPB leads to a discontinuous variation in the acceleration profile resulting in large jerk. Our system filters this effect out (see later) and, as a consequence, we can use LSPB to implement the Sample Trajectory Generator. For applications with many via-points, the easiest solution is to set up independent LSPB trajectories between adjacent via points and then to connect them one by one to form the whole sample trajectory sequence as shown in Eq. 25. In Section 4, we will give examples for this, as well as demonstrate the "filter" effect generated by the trained Modulation Function together with the Second-Order System.

3 The Whole Calculation Procedure

In the above sub-sections, we have disclosed all the key components of this method. Following Fig. 2, here we disclose the whole calculation procedure step by step, as follows.

(1) Use the LSPB method to generate the sample trajectory sequence $\mathbf{y}^*(t)$. Only positional boundary conditions, $(y_0,\ y_f)$ and possible via-points are used here, with the duration time T.
(2) Use the complete boundary conditions (y_0, y_f and $\dot{y}_0, \dot{y}_f$), and the duration time T to calculate the reference signal $r(t)$, as shown in Eqs. 16–18.
(3) Determine the parameters ω_n, C_1, C_2, l_1, l_2, t_m and finish the configuration of the Suppressing Window Function $v(t)$, and initialize the Gaussian Kernels Based Approximating Function $f(t)$ by all-zero weight vector $\mathbf{W}$ (see Eqs. 20 and 21). How to choose these parameters will be summarized later.
(4) Set a learning rate γ and start the iteration to adjust the weight vector $\mathbf{W}$, as shown in Eq. 22. For an accuracy-requiring application, we can choose smaller γ, and longer iteration times, and employ bigger M and bigger h. The influence of these parameters will be discussed in the Appendix.
(5) Perform the training iterations, where in every iteration, the output of the Second-Order System (see Fig. 2) will approach the sample trajectory $\mathbf{y}^*(t)$ more closely.

After this training procedure, the weight vector $\mathbf{W}$ contains the information from the sample trajectory sequence $\mathbf{y}^*(t)$. The whole architecture satisfies the given boundary conditions by itself and by now we have received the generated trajectory.

Please note, this method is a one-shot solution. It means that if we change any of boundary conditions (y_0, y_f and $\dot{y}_0, \dot{y}_f$), the position of the via-points, or the duration time T, we need to repeat the steps 4 and 5.

It seems that this method has many parameters and that it is quite complex. However, little tuning is required and once set up, we can use the parameters to deal with different and wide-ranging applications on a physical platform easily. Furthermore, we also can use this framework to generate different characteristic results for different application scenarios, just by adjusting some parameters. In the following Section, we will explain this by several cases.

4 Case Study: Application on Manipulators

Before discussing a robot application simulation, we present a simple 1-DOF case with detailed plots to show how our method works.

4.1 1-DOF Case

Figure 6 shows results from the 1-DOF case. This corresponds to a single coordinate in task space from a multi-DOF manipulator simulation study. In this case we used, $\omega_n = 50$, $\zeta = 1$, $\tau = 0.01$ s, $\gamma = 1$, and iteration time is 20. The kernel numbers are $M = 8$, and $h = 50$. Boundary conditions are $y(t_0) = 0.08$ m, $y(t_f) = 0.26$ m, $\dot{y}(t_0) = 0$ m/s, $\dot{y}(t_f) = 0.05$ m/s, $t_0 = 0$ s and $t_f = T = 2$ s.

According to the assigned boundary conditions, the reference signal $r(t)$ is plotted in Fig. 6a. The trained Gaussian kernels and the final modulation signal $f(t)$ are shown in Fig. 6b. The generated trajectories (red curves) are shown in Fig. 6c–f together with the sample trajectory (black curve). Please note here that the sample trajectory has velocity boundary conditions that equal zero, so that it is very easy to implement. The assigned boundary conditions are guaranteed by the architecture presented in this paper. From Fig. 6c–d we observe that the boundary conditions and the trajectories in the middle of the path match well to our inputs.

For comparison, in Fig. 7 we show a pair of results with identical parameters except final boundary velocity. All other parameters are as in Fig. 6. The final accuracy of these examples is listed in Table 1. Where, we define,

$$err_P = \left|\left(y_{target} - y_{real}\right)/y_{target}\right|. \tag{26}$$

and

$$err_V = \left|\left(\dot{y}_{target} - \dot{y}_{real}\right)/\dot{y}_{target}\right|. \tag{27}$$

We can see that the final accuracy is quite high.

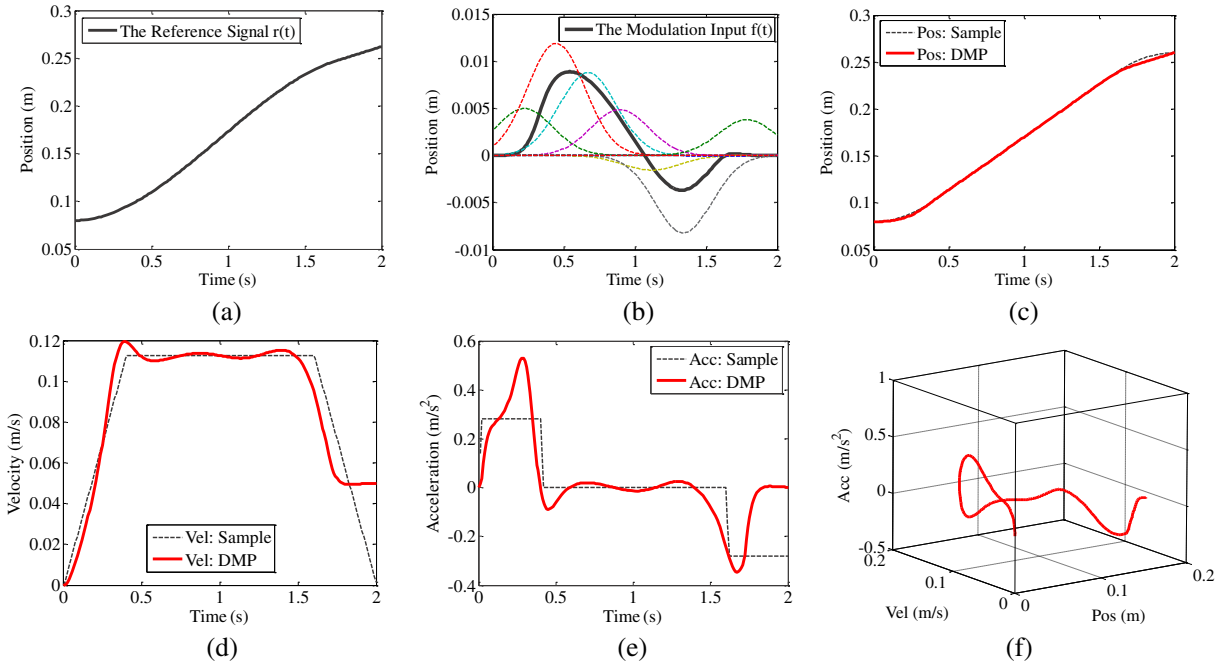

Fig. 6 1-DOF case. **a** Reference signal $r(t)$. **b** Gaussian Kernels and the final modulation signal $f(t)$. **c–e** Generated trajectory result and the provided sample trajectory information over time. **f** 3D phase track, "acceleration - velocity - position", of the generated trajectory

Analyzing the case shown in Fig. 7a and b, we can find an interesting phenomenon. Around 1.5 s to 2 s, the resulting trajectory deviates from the sample trajectory: it slows down and then approaches the assigned boundary condition. In fact, this phenomenon exhibits the autonomous

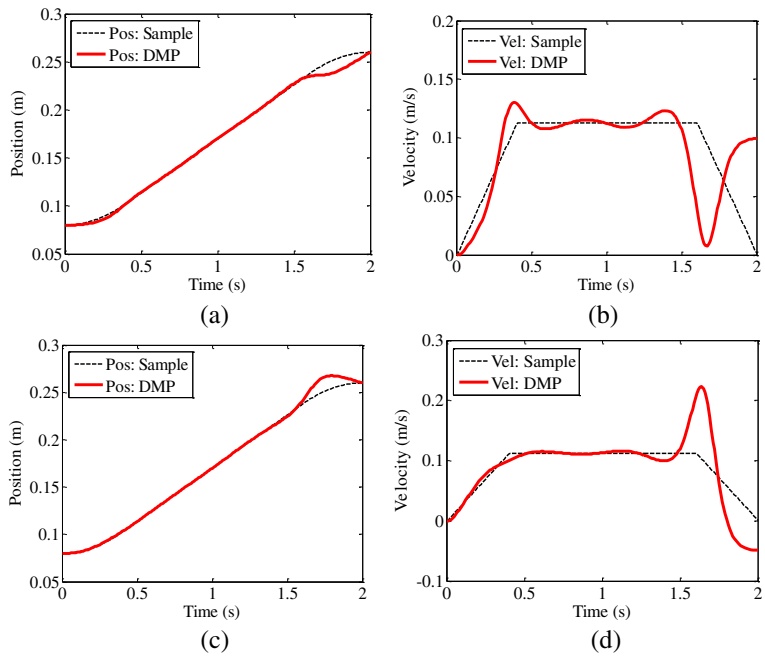

Fig. 7 Comparisons between two different final speed boundary conditions. **a**, **b** End speed is 0.1 m/s. **c**, **d** End speed is −0.05 m/s. The method shows a powerful autonomous capability to satisfy boundary conditions

Table 1 Accuracy of the resulting end-point

y_f(m),$\dot{y}_f$ (m/s)	err_P	err_V
0.26, 0.05	1.09×10^{-5}	1.3×10^{-3}
0.26, 0.1	7.37×10^{-5}	4.9×10^{-3}
0.26, -0.05	1.15×10^{-4}	1.6×10^{-2}

ability to adapt to the final boundary conditions, which is a property of our method. From a physics viewpoint, the two curves shown in Fig. 7b and their integral areas along the time axis must be equal to each other in order to achieve the same final position. As the assigned final speed $\dot{y}\left(t_f\right)$ is not zero, the near-to-the-end transition adjusts itself automatically to obey to this law. As during this period, $f(t)$ will be suppressed by $v(t)$ (see Eq. 20), $r(t)$ will govern the final result. In fact, when approaching an arbitrary assigned boundary condition, the "slowing down" and "reorienting" shown as Fig. 7a and b, lets the physical manipulator smoothly adapt to the required boundary conditions. The other two cases shown in Figs. 6 and 7 share the same principle. At this point, our method presents a "position and velocity" attractor, quite different from the original DMP framework presented in the literature [10–13].

Comparing the accelerations shown in Fig. 6e, we can find that the trained Modulation Function and the Second-Order System "filter out" the discontinuous variation brought from the LSPB based sample trajectory. As the acceleration starts from and ends at zero, the peak-to-peak value is bigger than the one from the given sample. This is also reasonable from a physical viewpoint but for real applications, peaks should be smaller than the limitations of the employed mechanical platform. If the peaks are too high to accept, we can reduce the sample acceleration at the start and end of the LSPB based sample trajectory.

A 3D "acceleration - velocity - position" phase track is shown in Fig. 6f. As mentioned before, the generated trajectory is very smooth, with low jerk.

In Table 2, we show the accuracy comparison between the parameters C_2 and t_m, for the cases in Figs. 6 and 7. If l_2 is bigger than $t_f - C_2$ this will yield better precision. The parameters affect the final accuracy. We give a summary in the Appendix.

Table 2 Accuracy comparison between $C_2$and t_m

	$t_m = t_f - 0.3$, ($\omega_n = 50$)	
	$C_2 = t_f - 0.28$	$C_2 = t_f - 0.48$
err_P	3.10×10^{-4}	4.04×10^{-7}
err_V	3.5×10^{-2}	7.44×10^{-5}
	$C_2 = t_f - 0.38$, ($\omega_n = 50$)	
	$t_m = t_f - 0.2$	$t_m = t_f - 0.4$
err_P	1.45×10^{-5}	8.65×10^{-6}
err_V	1.7×10^{-3}	9.12×10^{-4}

From the cases above, we can see that our method shows a powerful autonomous capability to satisfy boundary conditions with high precision. In the following multi-DOF case, this aspect will be more obviously demonstrated.

4.2 A Simple Case for a Multi-DOF Manipulator

So far our DMP based trajectory generator controls one DOF. For the N-DOF case, we can use N independent such trajectory generators in parallel, just sharing the same time base in order to synchronize their activities and to achieve coordinated motion. For instance, for a 6-DOF robot, we can employ six trajectory generators to represent all components (three for position and three for orientation) in Cartesian space. In order to control the manipulator, it is inescapable to transform the Cartesian trajectory into a joint space representation. This transformation is done by standard inverse kinematics, which we will not describe. For the following cases, the Robotics Toolbox V7.1 [28] was employed to aid our analysis.

Figure 8 shows the case of a 3-DOF manipulator, where the trajectory is generated in task space. The boundary conditions are: $\mathbf{X}(t_0) = [0.29, 0.08, -0.125]^T$, $\mathbf{X}(t_f) = [0.24, 0.26, -0.125]^T$, $\dot{\mathbf{X}}(t_0) = [0.02, -0.04, 0.08]^T$, and $\dot{\mathbf{X}}(t_f) = [-0.05, 0.1, -0.1]^T$.

The resulting trajectory, shown in Fig. 8a and b, matches to the target scenario described in Section 2.1. Figure 8c shows the velocity components. We can see that at the final end, the obtained velocities match to the assigned values. At the start, small jumps exist. This is due to the

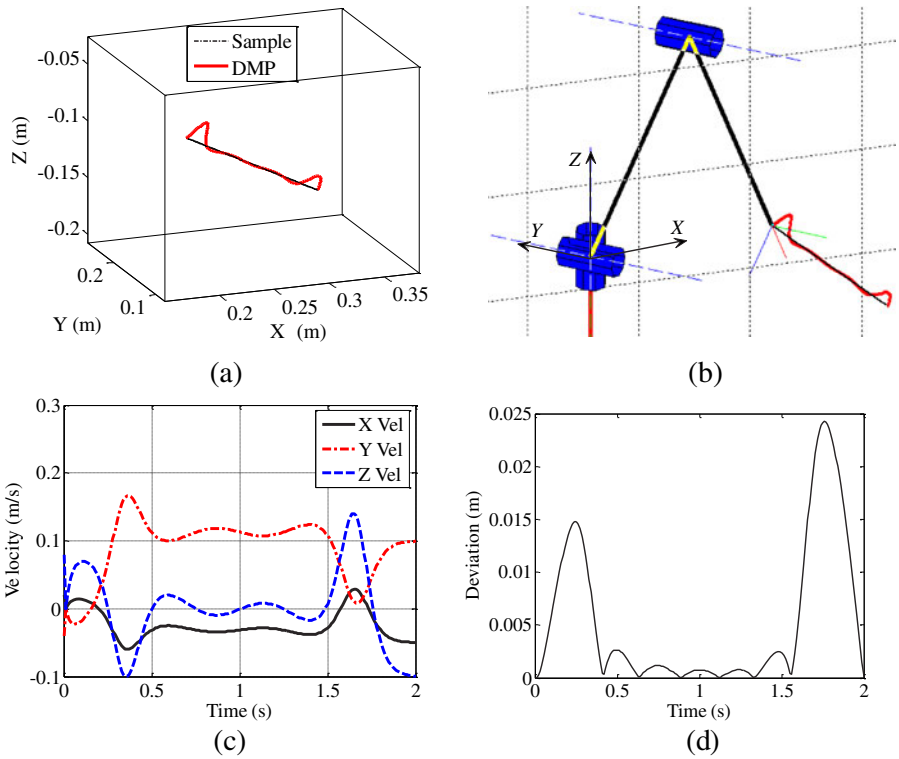

Fig. 8 Study of a 3-DOF manipulator. **a** The LSPB based sample and resulting trajectories in Cartesian space. **b** Plot of the robot model. **c** Velocity components. **d** Deviation between the sample and resulting trajectories

fact that, for the Second-order system shown in Fig. 3, it's zero initial conditions response follows:

$$\dot{y}(t) = \omega_n^2 t e^{-\omega_n t}. \tag{28}$$

Furthermore, we use a third-order polynomial to construct the Reference signal $r(t)$, and we cannot impose an initial acceleration on the conjoining (start) point.

However, non-zero start velocities would only be used for DMP joining and in this case the transition will be smooth due to matching the boundary conditions (end point to next start point) of two subsequent DMPs. Also, very short transients will always be smoothed by the real platform due to its intrinsic mechanical low-pass filter characteristics.

Note, the "deviation" plot shown in Fig. 8d does not mean "positional error". In this paper, we define "deviation" as $\left|\mathbf{y}^*(t) - \mathbf{y}(t)\right|$, to describe the distance between the corresponding track points (on the Sample trajectory and the resulting trajectory respectively, sharing the same time index). In the middle segment, the trajectory is close to the Sample. At the two ends, deviations are almost zero, which means that the boundary conditions are satisfied very well. The two peaks near the two ends describe the automatically generated transition zones (see Fig. 8a). Up to now, this simulation fulfills our target described in Section 2.1.

4.3 A Complicated Case for a Multi-DOF Manipulator

Our DMP based trajectory generator has the potential to deal with very complicated application cases. Here we show such a case and more characteristics of the method will be disclosed.

As shown in Fig. 9a, the manipulator needs to move along a 3D multi-segment path. Velocity vectors are assigned to start- and endpoints. The difference between cases shown in Fig. 9b, d, f and c, e, g is that they own different Gaussian kernel parameters. For the former one, $M = 4$, and $h = 10$; for the latter $M = 40$ and $h = 400$. The results show that with the increasing number of kernels the accuracy for both, position and velocity, increases.

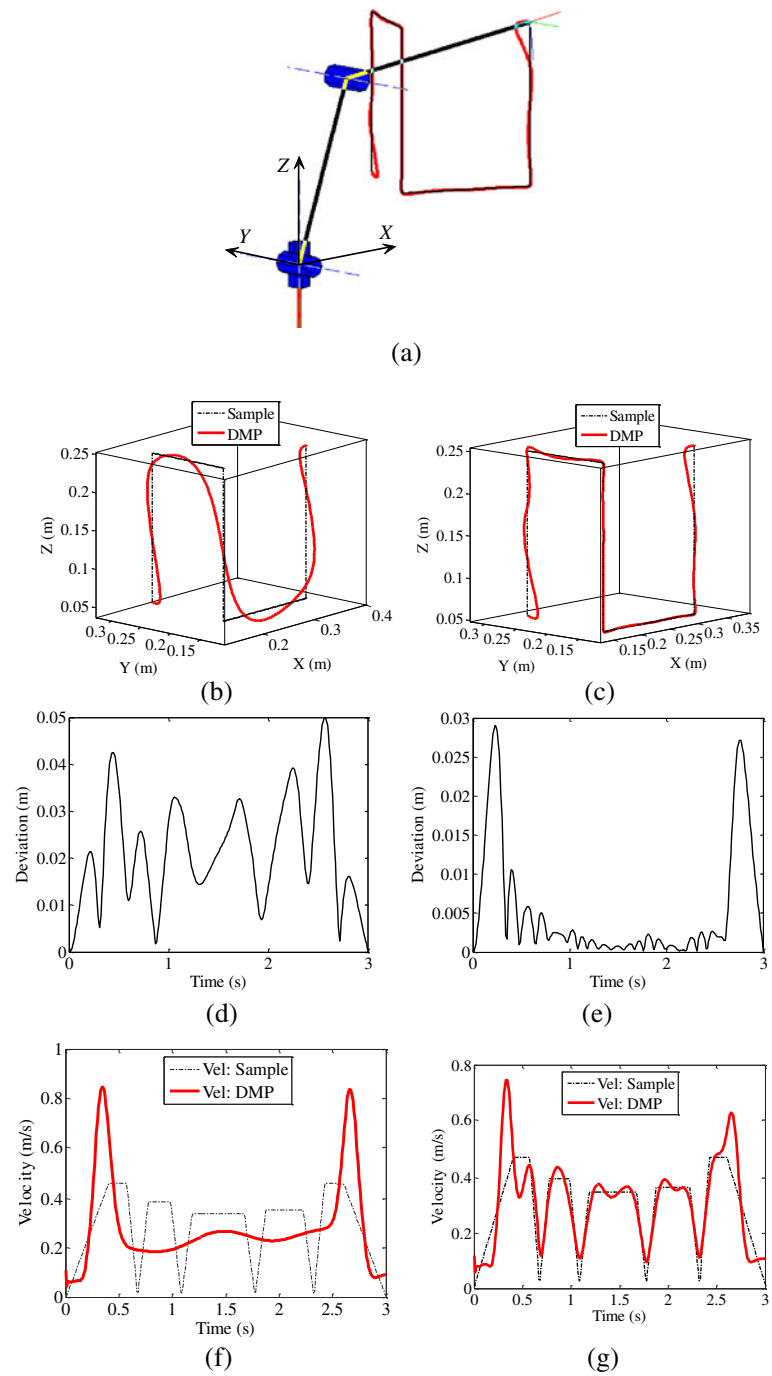

Fig. 9 A multi-segment trajectory generating and comparison case. Here, the robot needs to go along the straight lines defined by the multiple via-points one by one, with two-end none-zero velocity constraints. **b**, **d**, **f** Case uses fewer and wider Gaussian kernels: M = 4, and h = 10; **c**, **e**, **g** Case uses more and narrower Gaussian kernels: M = 40 and h = 400

In general, it is not feasible to let a robot track a sharp corner very fast. The widely employed traditional method is to set some transition zones to disconnect the connected lines and use arcs or higher order functions to smoothly connect them. The smallest curvature of the transition function will be dependent on the real platform's specification. Different from this, our solution deals with this problem more easily and naturally. As shown in Fig. 9g, the generated result

"slows down" and closely passes these sharp corners (via-points) autonomously. This method can learn well the information contained in the sample trajectory. This is a quite interesting advantage of this method.

Without loss of generality, in the cases shown in Fig. 9, the path segments are straight lines. We could in the same way also implement a different curve using another trajectory $\mathbf{y}^*$.

This case also shows another attractive potential of our method: we can use it to produce a complicated trajectory all in one time. In fact, this characteristic is totally different from all traditional solutions and quite attractive for complicated applications. For the case shown in Fig. 9, we don't need to produce five straight lines and six transitional curves (polynomials or arcs) separately and then connect them one by one, as the traditional approaches do.

It is also possible to employ some other optimized solutions to implement the Sample Trajectory Generator, because the following training course does not rely on any specific implementation.

Furthermore, as boundary conditions can be imposed on our trajectory generator we can produce several bigger sub-trajectories independently and connect them one by one to complete an even more complicated trajectory.

4.4 A Comparison Case – Spline-Base Trajectory Generation

In Fig. 10, two Spline-based trajectories are shown, using the same scenario as in Fig. 9. Different numbers of intermediate points are equally distributed between the via-points (corners). Position accuracy is – as expected – higher than for our method, but velocity in Cartesian space is heavily depending on the position interpolations and the corresponding time indexes of the intermediate points (note, we did not apply any optimizing technologies here). As disclosed here, in order to achieve a complex trajectory with an assigned speed profile, many intermediate points have to be defined [1] and the construction of the trajectory will become increasingly difficult.

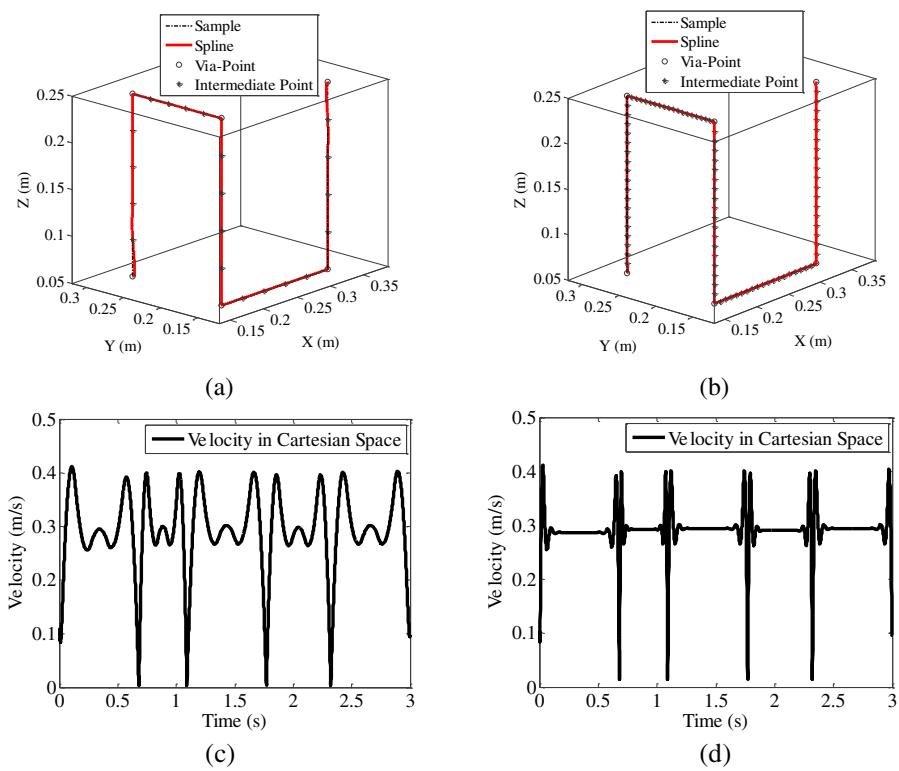

Fig. 10 Spline-based solution comparison cases, sharing the same scenario with the case shown in Fig. 9. **a**, **c** 4 intermediate points are equally distributed between two neighboring via points. **b**, **d** 18 intermediate points are equally distributed between two neighboring via-points. Please note, at the two ends, velocity conditions are $\dot{\mathbf{X}}(t_0) = [0.02, -0.1, -0.05]^T$ and $\dot{\mathbf{X}}(t_f) = [0.1, 0, 0]^T$, and zero speed are assigned at the remaining corner points

In the examples shown here the velocity profile is not really acceptable, especially case Fig. 10d. While this can be mended by using different support points for the spline, it is evident that this requires explicit considerations. Thus, there is a certain trade-off between position- and velocity-accuracy that needs to be explicitly dealt with when using splines. Different from that, our method *automatically* arrives at a very good compromise for position as well as velocity accuracy with only a moderate number of kernels (see. Fig. 9, right). Thus, without much effort our framework leds to quite flexible results and recalculation of a trajectory is easy once the method has been set up.

5 Conclusions and Application Potential

In this paper we presented a novel trajectory generator based on DMPs. For this, the original DMPs had been reformulated from a control theoretical viewpoint.

In the following we would like to compare our solution with other methods for trajectory generation. Most similarly there are Spline-based solutions [1, 4–9], against which our methods must "compete". The original DMP framework, on the other hand, contains several aspects which are different from our new approach (despite the fact that our approach was partly derived from DMPs). In Table 3, we give a summary comparison on the properties of these solutions some of which will be discussed in more detail next.

(1) Similar to splines but different from DMPs, we can define spatial and velocity constraints of the two ends of the trajectory support. The generated trajectory matches the assigned boundary conditions with high precision, with a definite temporal endpoint. As this design introduces the boundary conditions in an independent way, the path planning stage of a practical task can be simplified greatly.
(2) The generated trajectories are smooth: position and velocity profiles are continuous and differentiable, acceleration is continuous, and jerk is small.
(3) In our approach the generated trajectory sequence is passed through a built-in second-order system. As normally ω_n is low this acts as a low pass filter. Thus, the trajectory output will be easier to follow by motion control system [23], as long as the bandwidth of motion control system is higher than the one of our system. This is different from cubic or

Table 3 Property comparison between three kinds of trajectory generation technologies

Property	Splines	DMPs	Our Approach	Comment
Endpoint / end-velocity control	+/+	+/−	+/+	
Velocity / acceleration profile control	+/+	+/−	+/−	Splines need 5th order to control acceleration, too.
Chaining of primitives	+	−	+	
Via point control	+ (direct) + (indirect)	+ (indirect)	+ (direct) + (indirect)	Direct: by applying chaining. Indirect: by using kernel weights.
Built-in filter	−	+	+	
Time dependence	direct	indirect	direct	
Disturbance compensation	−	+	+	
Interpretability of coefficients	−	+	+	Interpretable means values have a human-understandable meaning.
Number of kernels/knots	4*N (3rd order)	N	N+4	
Number of free parameters	n.a.	2	6	
Phase stopping	−	+	−	
Generalization to different start and endpoints	−	+	−	
Bandwidth and error control against 2nd order system	−	+	+	Splines: Pure mathematical fitting. Behavior depends on motion servo.

higher order Spline-based trajectories, which do not match to the control system, even though they might have perfect mathematical fitting capabilities.

(4) Our method combines the sample trajectory $\mathbf{y}^*$ and boundary conditions, given independently, in a natural way. At the two ends a manipulator behaves in a very smooth way (see the ends of the paths shown in Figs. 7 and 9).

(5) Any reasonable required speed profile can be imposed on the generated trajectory.

(6) For multiple via-point applications we do not need to manually introduce (the traditionally used) zones for interpolating and connecting two adjacent segments. The method presented in this paper will generate smooth transition on its own, and the transition of the adjacent segments is also adjustable (see Section 2.5.3).

(7) Our approach presents a uniform framework to deal with simple as well as complex applications. The generated trajectory can be very complicated. This aspect is quite attractive, because when entering the path points, the velocity boundary conditions do not need to be considered, and thereby this simplifies the planning burden greatly.

(8) Depending on the application, we can implement one complicated trajectory passing closely all via-points at once, or generate several trajectory segments separately and then connect them by the pre-specified boundary conditions one by one. Using the latter approach, the robot can do its operations in position space with high accuracy. Note, we can impose velocity constraints on the end-positions to achieve also some dynamic tasks, e.g., to hammer, box, pitch, etc. This attractive property is quite obviously supported by the system presented in this paper.

(9) Our method provides strong flexibility in shaping the trajectory by ways of kernels (see Appendix A.3 and the middle of the paths shown in Fig. 9b and c).

Some limitations also presently exist:

(1) We have velocity peaks in the transition zones, caused by the quick change of the Suppressing Window Function, near C_1, C_2 (see Section 2.5.3, Fig. 9f). For real applications, the peak values should be below the limitation of the actuators of the employed manipulator. The solution to alleviate this problem is to use smaller accelerations for the LSPB based sample trajectory (see Section 2.6) or a longer whole time period T.

(2) When using a complete, complex trajectory via-points can be only passed closely; and the kernel numbers determines the actual distance errors (see the case shown in Fig. 9). For applications where via-points must be passed exactly, we need to generate sub-trajectories and connect them one by one.

Our method can be employed not only in task space, but also in joint space. In task space (Cartesian space), as the pre-planning work is more obvious, we can apply it as shown in Section 4. In joint space, without loss of generality, if one uses this method to directly generate the trajectories for each joint, it will also work. The principle is the same.

The often pursued combination of DMPs with imitation learning is also possible with this framework. For such an application, this method has the potential to allow for some dynamic tasks as mentioned above. For instance, a humanoid robot can imitate the movement profile and produce different kinds of end velocities.

Conventional industrial applications (e.g., [2, 29–36]) are certainly not in the focus of this paper. However, there is now quite a rising demand in industry for methods that are smoother, more easily adaptable, etc. This is where we see some future potential for our method.

All methods presented in this paper have been implemented on a simple robotic manipulator platform (Neuro-Robotics, Sussex) in our laboratory, where not only the key algorithm shown in this paper, but also more low-level driving and interfacing programs for the embedded control system of this platform have been implemented. In the literature [37], this method is employed. Several basic experiments and their explanations are given in our preliminary publication [38] of

this work. In summary, we believe that this novel trajectory generating technology exhibits great flexibility and applicability. Thus, we hope that our work presented in this paper will stimulate further DMP related research and development, and that this novel trajectory generating technology can be an alternative and widely employed not only in humanoid robots, but also in industry.

Acknowledgements The research leading to these results has received funding from the European Community's Seventh Framework Programme FP7/2007-2013 – Challenge 2 – Cognitive Systems, Interaction, Robotics – under grant agreement No 270273 – Xperience. It was also supported by the German BMBF BFNT 01GQ0810 project 3a of the University Göttingen.

The authors would also like to thank the anonymous reviewers who have made many valuable comments, which improved the paper.

Appendix: Summary of the Parameters

There are several free parameters used in this method. Here we give a summary on how to determine them for an application.

A.1 Dynamic Property Parameter: ω_n

In Eqs. 7 and 8, ω_n is related to this trajectory generator's dynamic property. As we know, the inertia of the mechanical components and its motion control system has limited response bandwidth. Thus, the trajectory output can be easily followed by the motion control system, as long as the bandwidth of the motion control system is higher than the one of our system. As a consequence of this the tracking error caused by the properties of the motion control system can be evaluated and remains well controlled [23].

For practical applications, we need to consider the following trade-off:

(1) Too small ω_n is meaningless because it will need a too long time to finish a trajectory, even though the results are very smooth (because it works as a low pass filter).
(2) Higher ω_n brings fast response times, but when limited by the parameters of a real platform, the generated result will challenge the motion control and actuation system.
(3) Founded by control theory, we just need to keep in mind that the employed ω_n should be lower than the one of the real platform. Then we can obtain an acceptable result (finite end-time, acceptable acceleration, etc). For all the cases shown in this paper we had $\omega_n = 50$.

A.2 Accuracy Related Parameters: t_m, C_1, C_2, l_1 and l_2

t_m is the junction moment of the polynomial segment and the linear segment, shown in Eq. 16. The time duration of the linear segment, $(t_f - t_m)$, is used to let the Second-Order System approximate the linear segment. It means that $(t_f - t_m)$ should be long enough to achieve an acceptable error.

Here we can arrive at a strategy: given err_P (see Eq. 26), we need to satisfy $t_f - t_m > t^*$, here, t^* is defined as follows,

$$err_p \approx e^{-\omega_n t^*}\left(1 + \frac{2}{\omega_n t^*}\right). \tag{29}$$

This relation is derived from the ramp input response (with zero state) of a second-order system, and please keep in mind that the steady-state error has been compensated by Eq. 17. By a numerical calculation program, Eq. 29 can be easily solved.

Please note that Eq. 29 is derived from a zero state and it has not taken into account the effect from the Suppressing Window Function. Equation 29 can be used as a basic reference to determine t_m. For a real application, to be on the safe side, the employed t^* should be bigger than the result shown in Eq. 29.

As disclosed in Eqs. 23, and 24, C_1, C_2, l_1 and l_2 determine the Suppressing Window Function. As mentioned there (Section 2.5.3), we need to ensure that $v(t)$ approaches zero at the start- and end-time as closely as possible in order to let the reference signal $r(t)$govern the Second-Order System Module's response to satisfy the given

boundary conditions. Generally, bigger l_1 and l_2 bring fast zero-approaching performance but induce higher velocity peaks as shown in Fig. 9f and g. C_1 and C_2 set the inflection points (see Fig. 5), and bigger values of them enforce the zero-approach. Furthermore, they are related to the transition zone of this method (e.g., see Figs. 1c and 8b).

In the cases shown above, we have presented suitable values of them, which can be directly used for many (also other) applications. All of them are related to the boundary condition accuracy of the trajectory generator presented in this paper.

A.3 Flexibility Related Parameters: M and h

M and h are related to the middle of the generated trajectory, and they will let the result exhibit different smoothness. Possible combinations are as follows:

(1) Large M and h means more and narrower basis functions. This should be employed for applications requiring high accuracy (e.g., the tracking accuracy between the given path is expected to be high).
(2) Small M and h: This should be employed, for example, for a humanoid robotic task, as the middle of the path accuracy is not a critical issue. In this way, the trajectory is smoother and "looks" more natural (human like).
(3) Small M and large h, this just brings some peaks to $f(t)$ and small ripples to the result and, thus, does not make sense. The generated trajectory will mostly follow $r(t)$, during the whole process.
(4) Large M and small h, the result is similar to (2), but one needs to spend more calculation resources.

In summary, concerning M and h, the choice is depending on the application requirements: if we need to follow a path as accurately as possible, we need to utilize narrower and more Gaussian kernels; if not, fewer and wider kernels are more suitable. We recommend (1) and (2) only, and they can be tuned/determined during a real task. Please compare cases shown in Fig. 9.

References

1. Siciliano, B., Sciavicco, L., Villani, L., Oriolo, G.: Robotics: Modelling, Planning and Control. Springer (2009)
2. Nystrom, M., Norrlof, M.: Path generation for industrial robots. Technical Report LiTH-ISY-R-2529, Department of Electrical Engineering, Linkoping University (2003)
3. Atkeson, C.G., Hale, J.G., Pollick, F., Riley, M., Kotosaka, S., Schaal, S., Shibata, T., Tevatia, G., Ude, A., Vijayakumar, S., Kawato, E., Kawato, M.: Using humanoid robots to study human behavior. IEEE Intell. Syst. **15**(4), 46–56 (2000)
4. Castain, R.H., Paul, R.P.: An on-line dynamic trajectory generator. Int. J. Rob. Res. **3**(1), 68–72 (1984)
5. Tondu, B., Bazaz, S.A.: The three-cubic method: an optimal online robot joint trajectory generator under velocity, acceleration, and wandering constraints. Int. J. Rob. Res. **18**(2), 893–901 (1999)
6. Thompson, S.E., Patel, R.V.: Formulation of joint trajectories for industrial robots using B-splines. IEEE Trans. Ind. Electron. **IE-34**(2), 192–199 (1987)
7. Chand, S., Doty, K.L.: On-line polynomial trajectories for robot manipulators. Int. J. Rob. Res. **4**(2), 38–48 (1985)
8. Taylor, R.H.: Straight line manipulator trajectories. IBM J. Res. Dev. **23**(4), 424–436 (1979)
9. Wada, Y., Kawato, M.: A theory for cursive handwriting based on the minimization principle. Biol. Cybern. **73**(1), 3–13 (1995)
10. Schaal, S., Peters, J., Nakanishi, J., Ijspeert, A.: Learning movement primitives. In: Proceedings of International Symposium of Robotics Research, pp. 561–572 (2003)
11. Ijspeert, A.J., Nakanishi, J., Schaal, S.: Movement imitation with nonlinear dynamical systems in humanoid robots. In: Proceedings of the IEEE International Conference on Robotics and Automation, pp. 1398–1403 (2002)
12. Peters, J., Schaal, S.: Reinforcement learning of motor skills with policy gradients. Neural Netw. **21**, 682–697 (2008)
13. Schaal, S., Mohajerian, P., Ijspeert, A.J.: Dynamics systems vs. optimal control - a unifying view. Prog. Brain Res. **165**(1), 425–445 (2007)
14. Vijaykumar, S., Schaal, S.: Locally weighted projection regression: an O(n) algorithm for incremental real time learning in high dimensional space. In: Proceedings of the International Conference on Machine Learning, pp. 1079–1086 (2000)
15. Ijspeert, A., Nakanishi, J., Schaal, S.: Learning attractor landscapes for learning motor primitives. In: Becker, S., Thrun, S., Obermayer, K. (eds.) Adv. Neural Inform. Process Syst., vol. 15, pp. 1547–1554 (2003)
16. Park, D.H., Hoffmann, H., Pastor, P., Schaal, S.: Movement reproduction and obstacle avoidance with dynamic movement primitives and potential fields. In: Proceedings of the IEEE-RAS Interna-

tional Conference on Humanoid Robots, pp. 91–98 (2008)
17. Hoffmann, H., Pastor, P., Park, D.H., Schaal, S.: Biologically-inspired dynamical systems for movement generation: automatic real-time goal adaptation and obstacle avoidance. In: Proceedings of the IEEE International Conference on Robotics and Automation, pp. 2587–2592 (2009)
18. Gams, A., Ude, A.: Generalization of example movements with dynamic systems. In: Proceedings of the IEEE-RAS International Conference on Humanoid Robots, pp. 28–33 (2009)
19. Pongas, D., Billard, A., Schaal, S.: Rapid synchronization and accurate phase-locking of rhythmic motor primitives. In: Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems, pp. 2911–2916 (2005)
20. Kober, J., Peters, J.: Learning new basic movements for robotics. In: Proceedings of the Autonome Mobile Systeme, pp. 105–112 (2009)
21. Kober, J., Mulling, K., Kromer, O., Lampert, C.H., Scholkopf, B., Peters, J.: Movement Templates for Learning of Hitting and Batting. In: Proceedings of the IEEE International Conference on Robotics and Automation, pp. 853–858 (2010)
22. Kulvicius, T., Ning, K., Tamosiunaite, M., Wörgötter, F.: Joining movement sequences: modified dynamic movement primitives for robotics applications exemplified on handwriting. IEEE Trans. Robot. **28**(1), 145–157 (2012)
23. Dorf, R.C., Bishop, R.H.: Modern Control Systems, 8th edn. Addison-Wesley (1998)
24. Pastor, P., Hoffmann, H., Asfour, T. Schaal, S.: Learning and generalization of motor skills by learning from demonstration. In: Proceedings of the IEEE International Conference on Robotics and Automation, pp. 763–768 (2009)
25. Nemec, B., Tamosiunaite, M., Wörgötter, F., Ude, A.: Task adaptation through exploration and action sequencing. In: Proceedings of the IEEE-RAS International Conference on Humanoid Robots, pp. 610–616 (2009)
26. Haykin, S.: Neural Networks: A Comprehensive Foundation, 2nd edn. Prentice Hall (1999)
27. Spong, M.W., Hutchinson, S., Vidyasagar, M.: Robot Dynamics and Control, 2nd edn. John Wiley & Sons (2004)
28. Corke, P.I.: A robotics toolbox for MATLAB. IEEE Robot. Autom. Mag. **3**, 24–32 (1996)
29. Kyriakopoulos, K.J., Saridis, G.N.: Minimum jerk path generation. In: Proceedings of the IEEE International Conference on Robotics and Automation, pp. 364–369 (1988)
30. Tarn, T.J., Xi, N., Bejczy, A.K.: Motion planning in phase space for intelligent robot arm control. In: Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems, pp. 1507–1514 (1992)
31. Piazzi, A., Visioli, A.: Global-minimum-jerk trajectory planning of robot manipulators. IEEE Trans. Ind. Electron. **47**(1), 140–149 (2000)
32. Piazzi, A., Visioli, A.: An interval algorithm for minimum-jerk trajectory planning of robot manipulators. In: Proceedings of the IEEE Conference on Decision and Control, pp. 1924–1927 (1997)
33. Nguyen, K.D., Ng, T.C., Chen, I.M.: On algorithms for planning S-curve motion profiles. Int. J. Adv. Robot. Syst. **5**(1), 99–106 (2008)
34. Constantinescu, D., Croft, E.A.: Smooth and time-optimal trajectory planning for industrial manipulators along specified paths. J. Robot. Syst. **17**(5), 223–249 (2000)
35. Shin, K.G., McKay, N.D.: Minimum-time control of robot manipulators with geometric path constraints. IEEE Trans. Automat. Contr. **30**(6), 531–541 (1985)
36. Bobrow, J.E., Dubowsky, S., Gibson, J.S.: Time-optimal control of robotic manipulators along specified paths. Int. J. Rob. Res. **4**(1), 3–17 (1985)
37. Aksoy, E.E., Abramov, A., Dellen, B., Dörr, J., Ning, K., Wörgötter, F.: Unsupervised recognition and classification of manipulations: grouping actions and objects. Int. J. Rob. Res. **30**(10), 1229–1249 (2011)
38. Ning, K., Kulvicius, T., Tamosiunaite, M., Wörgötter, F.: Accurate position and velocity control for trajectories based on dynamic movement primitives. In: Proceedings of the IEEE International Conference on Robotics and Automation, pp. 5006–5011 (2011)
39. Marquez, H.J., Damaren, C.J.: On the design of strictly positive real transfer functions. IEEE Trans. Circuits Syst. Fundam. Theory Appl. **42**(4), 214–218 (1995)
40. Ioannou, P., Gang, T.: Frequency domain conditions for strictly positive real functions. IEEE Trans. Automat. Contr. **32**(1), 53–54 (1987)
41. Huang, C.-H., Ioannou, P.A., Maroulas, J., Safonov, M.G.: Design of strictly positive real systems using constant output feedback. IEEE Trans. Automat. Contr. **44**(3), 569–573 (1999)
42. Sane, H.S., Bernstein, D.S.: Asymptotic disturbance rejection for hammerstein positive real systems. IEEE Trans. Control Syst. Technol. **11**(3), 364–374 (2003)